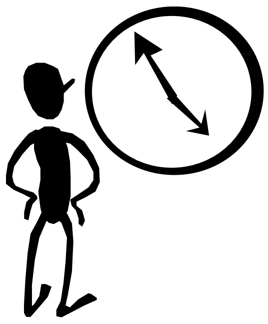
Frequency Measurement Using Input Capture with the Cerebot MX7cK™

Revision: 07 Dec 2017 (JFF)
Richard W. Wall, University of Idaho, rwall@uidaho.edu

www.digilentinc.com
1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Project 10: Frequency Measurement Using Input Capture



Purpose

The purpose of this project is to use the PIC32 input capture to measure frequency. Using such a frequency measurement will allow us to determine the speed of a DC motor. This project uses a period measurement to determine frequency. Project 9 should be completed prior to completing Project 10.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB ® X IDE
3. Understanding of PWM principles
4. Concepts of frequency measurements (bad link!) ☹
5. Use of logic analyzer (7 channels will be simultaneously monitored)

Equipment List

1. Digilent Cerebot MX7cK processor board with USB cable
2. Microchip MPLAB ® X IDE
3. Microchip MPLAB ® XC32 Compiler
4. Digilent PmodCLP Parallel Character LCD
5. Digilent H-Bridge driver PMod
6. Digilent DC Motor
7. 8 Channel Logic analyzer (Digilent Analog Discovery)

Software Resources

1. XC32 C/C++ Compiler Users Guide
2. PIC32 Peripheral Libraries for MBLAB C32 Compiler
3. PIC32 Family Hardware Reference Manual Section 16 Output Compare
4. Cerebot MX7cK Board Reference Manual
5. MPLAB ® X Integrated Development Environment (IDE)

References

1. PIC32 Input Capture
2. Voltage to Frequency Converter
3. Using PWM to Generate Analog Output
4. C Programming Reference

Analog to Digital Conversion

Project 9 investigated methods of using a microprocessor to generate a variable amplitude output. In Project 10, we will look at one method of using a microprocessor to measure a variable-valued input. The PIC32 processor has 16 analog inputs channels for a 10-bit ADC. However, we will use the DC motor tachometer to measure the analog speed value. Considering that the motor speed is controlled by the average applied voltage, the tachometer represents a voltage to frequency converter. Various silicon manufacturers offer electronic devices that are used to generate a pulse waveform whose frequency is a function of the voltage applied to the sensor input. Using the frequency of a digital pulse wave to represent an analog signal has the advantages of higher signal to noise ratio and reduces sensitivity to circuit impedance.

Project 9 implemented open loop speed control of a DC motor. This control approach depends upon the proportionality of the motor speed to the applied voltage. Project 10 is designed to test the linearity of open loop speed control by providing a tachometer to measure the rotor shaft speed. The hardware requirements for Project 9 are the same as for Project 10. Details concerning the hardware configuration for this project are provided in Appendix A through Appendix E. The DC motor used on the Digilent Cerebot MX7cK Project System pictured in Appendix D has two Hall Effect sensors that provide a pulse each time the DC motor shaft makes a revolution. The two sensors are physically oriented around the motor rotor shaft such that they generate two digital signals that oscillate at the same frequency but are out of phase by 90 degrees. The SA and SB PmodHB5 Hall Effect sensor outputs form a quadrature encoder that allows both the speed and direction of rotation to be measured. We are only concerned with measuring the motor speed in Project 10.

The speed of the geared output shaft is reduced by the ratio of 53 to 1 to provide greater mechanical torque. The speed of the motor will be measured by determining the frequency of the digital pulse wave generated by the Hall Effect sensor.

Fundamentals of Frequency Measurement

As with the measurement of any data, we are concerned with three parameters: accuracy, resolution, and precision. The accuracy of a frequency measurement is dictated, to a large extent, by the accuracy of the reference crystal or oscillator. For this project, we depend on the accuracy of the oscillator populated on the Cerebot MX7cK processor board. The measurement precision is a metric applied to the repeatability of a measurement. Resolution is the smallest interval between two measurements. Accuracy is the difference between the measured value and an absolute standard (the "truth"). Measurement precision and resolution are determined by the processing of the measured data. The subject of frequency measurement resolution is of great interest and is addressed in greater detail in the following paragraphs.

There are two methods of estimating the frequency of the tachometer. The number of signal transitions (rising, falling, or both) in a fixed time interval can be counted, or the time interval between two consecutive signal transitions can be measured using a timer. Measuring frequency by counting transitions over short intervals is more accurate for signals when the frequency of the signal being measured is high relative to the observation interval. This results in many hundreds or even thousands of cycles being counted during the measurement interval, thus providing a high frequency measurement resolution.

Measuring the period is generally more precise for relatively low frequency signals. The resolution is provided by the number of timer counts between input signal transitions. The measurement period now depends of the frequency of the input signal and, for very low frequency signals, may result in excessively long delays between measurement updates.

There are methods to mitigate the effects of long measurement periods for low frequency signals. One method is to use frequency multiplication with a phased locked loop (PLL) circuit. Most modern microcontrollers and microprocessors use frequency multiplication so that the core speed of the processor is many times the processor crystal frequency. The PIC32 processor family uses a phase locked loop multiplier to set the core frequency by a parameter in the config_bits.h file. Another method for handling the measurement of signals with long periods is to use clocks that run at very low frequencies.

For Project 10, you will measure the speed of the motor rotor shaft by computing the period between two successive pulses of the motor tachometer by capturing a timer count on each falling transition. In order to achieve precise measurements the period must be measured with high resolution. This requires that the timer be counting at a much higher rate than the period of the of the tachometer signal. The precision of the frequency estimate is preserved by inverting the period using floating point variables or fixed point math divide algorithms.

The challenge is to select the timer input clock rate that will provide adequate period accuracy across the range of possible tachometer frequencies. The maximum number of counts is 65535 timer increments. Using a peripheral bus clock (PBCLK) frequency of 10 MHz and a timer pre-scale value of 256, the minimum measurable frequency is 0.596 Hz but will require 1.68 seconds to get a measurement update. The maximum measurable frequency, corresponding to one timer count, is 39 kHz, but the resolution degrades to 39 kHz. A signal frequency of 197.6 Hz will result in a 1 Hz resolution (Equation 1).

Granularity is a relative term that relates to the coarseness of a measurement. Higher granularity means lower resolution[1]. From the derivation shown in Appendix F, resolution of the frequency measurement when using a timer to measure the signal period can be determined using the equation:

$$\Delta F = \frac{\mathrm{T3_{PD}}}{T_{SIGNAL}} \cdot \mathrm{F_{SIGNAL}} = \mathrm{T3_{PD}}\,\mathrm{F_{SIGNAL}}^2 \qquad \text{Eq. 1}$$

where $F_{SIGNAL} = 1/T_{SIGNAL}$ and $T3_{PD} = T3_{PRESCALE}/PBCLK$

Eq. 1 shows that, for a fixed timer clock frequency, the granularity of the frequency measurement**,** ΔF, increases as the square of the input frequency and has dimension of Hz. As Figure 1 illustrates, a higher input frequency will result in a measurement with higher granularity for a given measurement interval. Better resolution (finer granularity) is achieved by increasing the clock frequency or reducing input frequency.

Measuring a signal frequency by computing the inverse of the signal period will generate a new measurement update at the signal frequency. Provided that the input signal has a 50% duty cycle, we can get two frequency measurements per cycle of the input signal by determining the period when the input is high and the period when it is low. Measurement noise can be reduced by averaging multiple period measurements.

The speed of the motor is proportional to the frequency of the signal generated by the motor tachometer. The motor that we are using has a reduction gear with a ratio of 53:1. Using a 10V motor supply, the maximum motor speed will be approximately 525 revolutions per second (RPS) or 594 RPM on the output of the gear head. Due to motor power limitations and gear head friction, the minimum motor speed is approximately 79.5 RPS which is equivalent to 90 RPM as measured on the gear-head output shaft.

[1] Avoid using these terms!

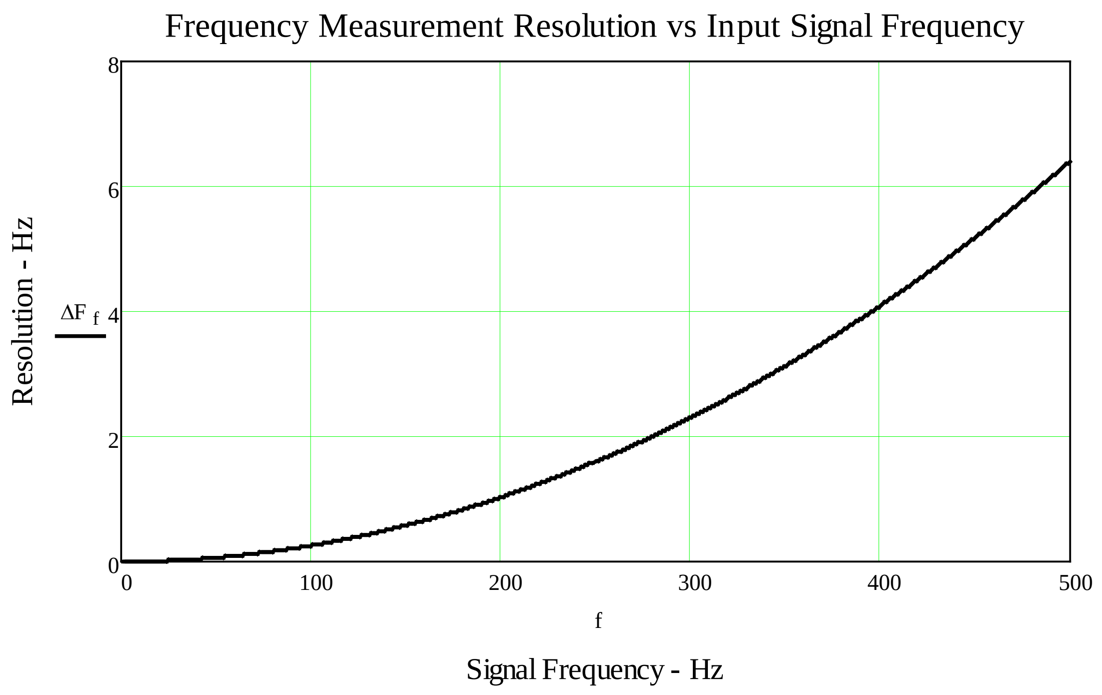


Figure 1. Plot of frequency measurement resolution for a fixed clock frequency.

Timer Setup for PWM and Input Capture

For this project, we will use two PIC32 timers: one for generating the PWM output that controls the motor speed (Timer 2), and one that will be used to capture the time when the tachometer signal transitions from low to high or high to low (Timer 3). The reason for using two timers is that the period register of the two timers are set to different values in order to achieve different design objectives. The period register of the timer used for generating the PWM signal is set to give a particular PWM cycle frequency. The period register of the timer used for measuring the period of the tachometer signal is set to the maximum value to allow the period measurement of an input signal over a wider range of frequencies. The lower the frequency on the input signal, the more timer counts that will be measured. The period measurement will be incorrect if the timer resets after reaching the terminal count more than once between input signal transitions. Hence, the period register for the timer that is used for timer capture will be set to its maximum value, 65536.

The ReadCapture() function from the peripheral library uses a pointer to an array of unsigned integers (32 bits), even though the timer is only 16 bits. In order to correctly compute the number of timer increments in the event of timer rollover between two samples we must treat the values in the array as *unsigned short int* data types. For example, consider the case when the first captured timer value is 0xFE3E (65086 in decimal) and the second 0x0096 (150). The result of (0x0096 – 0xFE3E) is 0x0258 (600) – the true interval in timer increments. A more thorough discussion of time interval issues is provided in the section titled, "*Handling Timer Rollover for Hardware Assisted Delays*" in Project 2.

PIC32 Software for Timer 2 Setup for PWM and Timer Interrupt

Timer 2 will serve two functions: it will be used to generate the PWM signal and it will generate an interrupt once each millisecond. The one millisecond interrupt will be used for hardware timing. Configure Timer 2 for operations as follows:

1. Configure Timer 2 from 1ms period
    a. *OpenTimer2(T2config_bits, PR2_value);* where (PR2_value + 1) is the period register value computed for the PWM cycle frequency.
        i. *T2config_bits = (T2_ON* | T2_Ps_1_1 | T2SOURCE_INT)
        ii. PR2 = (10000-1) for PBCLK = 10MHz

2. *Configure Output Compare for 1ms PWM cycle period*
    a. *OpenOCx(OC_configure_bits, nOCxRS, nOCxR);* where nOCxRS is the initial value written to the OCxRS register and nOCxR is the initial value written to the OCxR register. "x" designates the specific OC register being configured in the range of 1 through 5.
        i. *OC_configure_bits* (b1 | b2 | b3 | b4)
            1. b1 = OC_ON //Enables the Output compare processor resource
            2. b2 = OC_TIMER_MODE16 // Timer uses 16 bit mode
            3. b3 = OC_TIMER2_SRC // Selects Timer 2 as input timer
            4. b4 = OC_PWM_FAULT_PIN_DISABLE // Don't use fault pin
        ii. Output Compare Registers
            1. *nOCxRS* – This is a constant or a variable that specifies the initial PWM compare value
            2. *nOCxR* – This is a constant or a variable that specifies the next PWM compare value

        Refer to the Section 13 of the C32 Peripheral Library Guide for option details.
    b. Example: *OpenOC3(OC_configure_bits, nOC3RS, nOC3R);*

3. Timer 2 interrupt using peripheral library macro functions
    a. *mT2SetIntPriority(2);* *// Set Timer 2 interrupt group priority level 2*
    b. *mT2SetIntSubPriority(1);* *// Set Timer 2 interrupt subgroup priority* 1
    c. mT2IntEnable(1); // Enable T2 interrupts
    d. Alternatively, using the C32 Peripheral Library function:
    ConfigIntTimer2(T2_INT_ON | *T2_INT_PRIOR_2* | *T2_INT_SUB_PRIOR_1);*

4. Timer 2 ISR code

```
void __ISR( _TIMER_2_VECTOR, ipl2) T2Interrupt(void);
{
        LATBINV = LEDA;      // Toggle LED A – instrumentation
        mT2ClearIntFlag();   // Clear Timer 2 interrupt flag
        // INTClearFlag(INT_T2);      // Alternate peripheral library function
}
```

PIC32 Software Timer Setup for Input Capture

Timer 3 Interrupt Setup

Timer 3 will be used exclusively for a 16 bit input timer capture.

1. Configure Timer 3 to use the PBCLK and a pre-scale value of 256. Set Timer 3 PR3 register to 0xFFFF for maximum interval.
    a. *OpenTimer3(T3_ON* | *T3_PS_1_256* | *T3_SOURCE_INT, 0xFFFF);*
    b. *mT3SetIntPriority(2); // Set Timer 3 interrupt group priority 2*
    c. *mT3SetIntSubPriority(2); // Set Timer 3 interrupt subgroup priority 2*
    d. *mT3IntEnable(1); // Enable Timer 3 interrupts*
    e. Alternate to steps b through d using the C32 Peripheral Library function:
    ConfigIntTimer3(T3_INT_ON | *T3_INT_PRIOR_2* | *T3_INT_SUB_PRIOR_2);*

2. Timer 3 ISR

```
void __ISR( _TIMER_3_VECTOR, ipl2) T3Interrupt(void)
{
    LATBINV = LEDC;          // Toggle LEDC  - Timing instrumentation
    mT3ClearIntFlag();       // Clear Timer 3 interrupt flag –
    // INTClearFlag(INT_T3); // Alternate peripheral library function
}
```

Input Capture Initialization

1. Set motor Hall effect outputs SA (RD3) and SB (RD12 - IC5) as inputs. MTR_SA is set as an input but is not used for this application because RD3 is not a designated input capture pin. (Note: MTR_SA and MTR_SB must be defined appropriately.)
    a. *PORTSetPinsDigitalIn(IOPORT_D, (MTR_SA* | *MTR_SB));*
2. Clear pending input capture interrupts
    a. *mIC5ClearIntFlag();*

3. Use input capture channel 5 for the following options. An interrupt will be generated on each falling edge of the input signal.
void OpenCapture5(c1 | *c2* | *c3* | *c4* | *c5* | *c6* | *c7); // Note the logical OR!!!*[2]

a.	c1 = IC_ON	// Enable input capture
b.	c2 = IC_CAP_16BIT	// Capture a 16 bit timer count
c.	c3 = IC_IDLE_STOP	// Stop input capture during debug
d.	c4 = IC_FEDGE_FALL	// Initial capture on falling edge
e.	c5 = IC_TIMER3_SRC	//Use Timer 3 as time to capture
f.	c6 = IC_INT_1CAPTURE	// Generate interrupt on each capture
g.	c7 = IC_EVERY_ FALL_EDGE	// Capture time on each falling edge

[2] The sample code on p. 176 of the Peripheral Library Guide incorrectly uses logical AND. ☹

4. Configure the Input Capture interrupt as follows
    a. ConfigIntCapture5(ic1 | ic2 | ic3);
        i. ic1 = IC_INT_ON // Enable input capture interrupt
        ii. ic2 = IC_INT_PRIOR_3 // Set priority for level 3
        iii. ic3 = IC_INT_SUB_PRIOR_0// Set sub priority for level 0
    b. Example:
    ConfigIntCapture5(IC_INT_ON **|** *IC_INT_PRIOR_3* **|** *IC_INT_SUB_PRIOR_0);*

Input Capture ISR Example

```
void __ISR( _INPUT_CAPTURE_5_VECTOR, ipl3) Capture5(void)
{
        static unsigned int con_buf[4];         // Declare an input capture buffer
        // Declare three time capture variables:
        static unsigned short int t_new;        // Most recent captured time
        static unsigned short int t_old = 0;    // Previous time capture
        static unsigned short int time_diff;    // Time between captures

        LATVBINV = LEDD                         //Toggle LEDD on each input capture interrupt
        ReadCapture5(con_buf);                  // Read captures into buffer
        t_new = con_buf[0];                     // Save time of event
        time_diff = t_new – t_old;              // Compute elapsed time in timer "ticks"
        t_old = t_new;                          // Replace previous time capture with new
        // Compute motor speed in RPS (revolutions per second) and save as global variable[3]
        // Details left as an exercise for the Reader ☺
        mIC5ClearIntFlag();             // Clears interrupt flag
        // INTClearFlag(INT_IC5);       // Alternate peripheral library function
}
```

Project Tasks

The objective for this project is to implement a motor speed tachometer by measuring the frequency of the motor shaft sensor.

1. Develop code that generates the PWM signal
    a. Initialize the PWM duty cycle to 40%
    b. Toggle LEDA each Timer 2 interrupt
    c. Set LEDB at the start of a CN ISR and clear LEDB at the end of the CN ISR
    d. Toggle LEDC each Timer 3 interrupt
    e. Toggle LEDD each input capture interrupt

[3] Compute the motor speed as a moving average of at least ten samples.

2. Write and verify the C code to complete the button control and LCD display design specified below.

3. Connect the logic analyzer probes as follows to demonstrate that the PWM output and Timer 2 interrupts continue to function while the CN interrupt is being served.
   a. Channel 0: LEDA – Measures Timer 2 interrupt timing
   b. Channel 1: LEDB – Measures the ISR duration that detects a button press
   c. Channel 2: LEDC– Measures Timer 3 interrupt timing
   d. Channel 3: LEDD – Measures the Input capture timing
   e. Channel 4: PmodHB5_EN – PWM output signal
   f. Channel 5: PmodHB5_SA – Motor tachometer Phase A
   g. Channel 6: PmodHB5_SB – Motor tachometer Phase B

4. Capture the logic analyzer screen for the four PWM duty cycle settings specified in Table I. (See item three under project testing.)

Table I. Button controlled PWM duty cycle

BTN2	BTN1	PWM
OFF	OFF	40%
OFF	ON	65%
ON	OFF	80%
ON	ON	95%

Project Specifications

1. Operating Requirements:
   a. Foreground operations: (operations implemented from ISR)
      i. Toggles LEDA in the Timer 2 ISR at the 500 Hz rate (LEDA is toggled at one half of the PWM cycle frequency)
      ii. Detect button uses the change-notice interrupt similar to the Project 5 implementation.
         1. The CN interrupt is to use a 20 ms software delay for switch debounce
         2. LEDB is turned on for the duration of the CN ISR.
         3. Sets the PWM output as a function of the states of buttons BTN1 and BTN2 as specified in Table I
      iii. Toggles LEDC for each Timer 3 interrupt (1.68 second period) (the maximum time before a Timer 3 roll-over)
      iv. Toggles LEDD each time the input capture ISR is serviced (the speed of the DC motor)

      v. Writes the PWM percent duty cycle to line 1 of the LCD whenever the buttons change the %PWM (Hint: Generate a function the positions the LCD cursor to a fixed position. Then write a fixed length character string to update the element of the display.)
   b. Background operation:
      i. Updates the motor speed measurement (RPM) on line 2 of the LCD display at the rate of once each 100ms as determined by the software delay function, "DelayMs", that can be preempted
      ii. Clear only line 2 of the LCD and display the measured revolutions per second as RPS = xxx.xx
      iii. The LCD updates must be protected from CN interrupts

2. Software Organization:
   a. "main"
      i. Calls application initialization function that completes the following:
         1. Cerebot MX7ck board configured inputs for BTN1 and BTN2
         2. Cerebot MX7ck board configured outputs for PmodSTEP LEDA through LEDD
         3. Timer 2 to generate an interrupt each millisecond.
         4. The PWM output channel as follows
            a. Use output compare 3 (OC3)
            b. Uses Timer 2 for the time base for the output compare
            c. PWM cycle frequency of 1000 Hz
         5. Timer 3 to generate an interrupt every 1.68 seconds. (Prescale of 256 and a period of 0xFFFF.)
         6. Input Capture (See Input Capture ISR Example)
            a. Use Input Capture 5
            b. Trigger on each falling edge of the input pin IC5 (RD12)
         7. Use a 16 bit timer
            a. Use Timer 3 as time reference
         8. Initialize a CN interrupt for button status detection at priority 1 and the sub priority level 0.
         9. LCD initialization with a function added to position the LCD cursor at any position. (Refer to Project 6 for LCD programming)
      ii. Executes a while(1) loop (background operations)
         1. Updates the motor speed measurement (RPM) on line 2 of the LCD display at the rate of once each 100ms as determined by the software delay function, "DelayMs", that can be preempted
         2. Clear only line 2 of the LCD and display the measured revolutions per second as RPS = xxx.xx
         3. The LCD updates must be protected from CN interrupts
   b. Button Detect ISR (refer to Project 5)
      i. Sets LEDB on entry and clears LEDB on exit
      ii. Remove button contact bounce with 20ms software delay
      iii. Reads button state
      iv. Decodes buttons and sets PWM in accordance with Table I
      v. Sets the motor PWM
      vi. Updates only line 1 of the LCD reporting the % PWM using the format: PWM = ##%

vii. Clears CN interrupt flag

c. Timer2 ISR
   i. Toggles LEDA
   ii. Clears T2 interrupt flag
d. Timer 3 ISR
   i. Toggle LEDC
   ii. Clear T3 interrupt flag
e. Input capture ISR
   i. Clears input capture interrupt flag.
   ii. Toggles LEDD
   iii. Computes the time interval since the last input capture interrupt
   iv. Computes a moving average of motor speed – RPS (global variable)[4]

Project Testing

The DC motor control block diagram is shown in Appendix B, The Hall Effect device output labeled SB is connected to the IC5 / RD12 pin. Consequently, we will be configuring the PIC32 to use input capture channel 5. Refer to the *OpenCapture* topic in the C32 peripheral library for the setting of the input capture module.

1. Connect the test equipment to the Cerebot MX7cK using a logic analyzer as follows:
   a. Connect the logic analyzer CH 0 through CH 3 probes to the test points for LEDA through LEDD on the PmodSTEP.

   b. Connect logic analyzer CH 4 probe to the test point for the PWM EN pin on the PmodHB5.

   c. Connect logic analyzer probes for CH 5 and 6 to the test points for the SA and SB pins on the PmodHB5

.

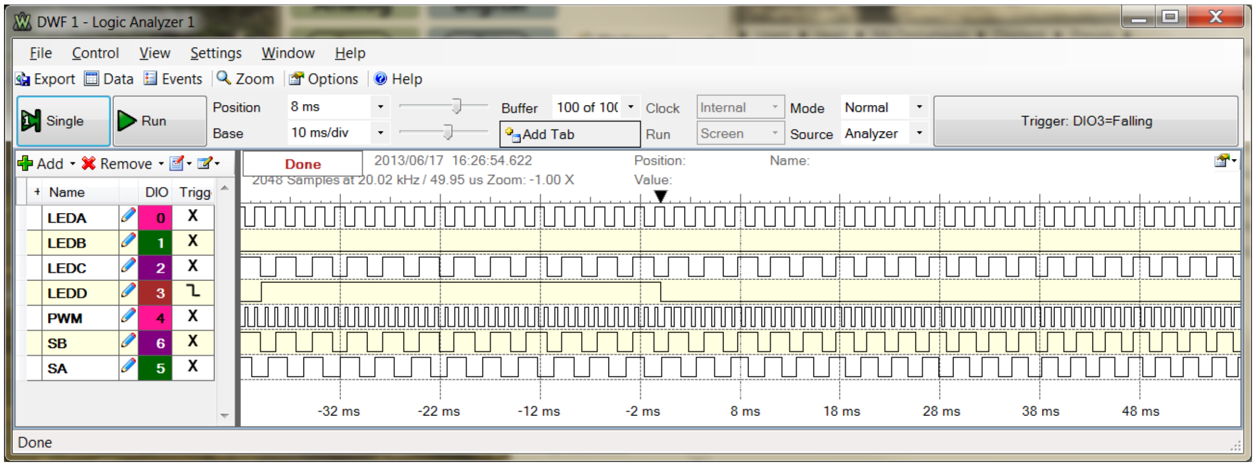
2. Run the Project 10 program to record the measurements indicated in Table II for the case when the PWM is set to 65%. ***Measure and record the motor supply voltage***.

Table II.

Instrumentation TP	Function	Measurement – Hz
PmodSTEP LEDA	PWM cycle frequency (Timer2)	
PmodSTEP LEDB	Length of CN interrupt	
PmodSTEP LEDC	Input capture timer frequency (Timer3)	
PmodSTEP LEDD	Input capture frequency	
PmodH5 EN	PWM waveform	
PmodH5 SA	Tachometer frequency	
PmodH5 SB	Tachometer frequency	

3. Provide a screen capture of the seven signals instrumented to complete Table II. An example plot is shown in Figure 2. (Note: LEDC and LEDD were swapped below.)

[4] Minimum of ten samples.



Position 8 ms
Base 10 ms/div
Buffer 100 of 100
Clock Internal
Run Screen
Mode Normal
Source Analyzer
Trigger: DIO3=Falling

Done 2013/06/17 16:26:54.622
2048 Samples at 20.02 kHz / 49.95 us Zoom: -1.00 X
Position:
Value:
Name:

Name	DIO	Trigg
LEDA	0	X
LEDB	1	X
LEDC	2	X
LEDD	3	⌊
PWM	4	X
SB	6	X
SA	5	X

Done

Figure 2. Screen capture of the seven monitored signals.

4. Complete Table III and graph the frequency measured at the PmodHB5 SB pin versus the percent PWM duty cycle specified in Table I. A sample plot of this data is provided in Figure 3.

Table III. Motor Tachometer frequency for button controlled PWM duty cycle

BTN2	BTN1	PWM	SB – Hz
OFF	OFF	40%	
OFF	ON	65%	
ON	OFF	80%	
ON	ON	95%	

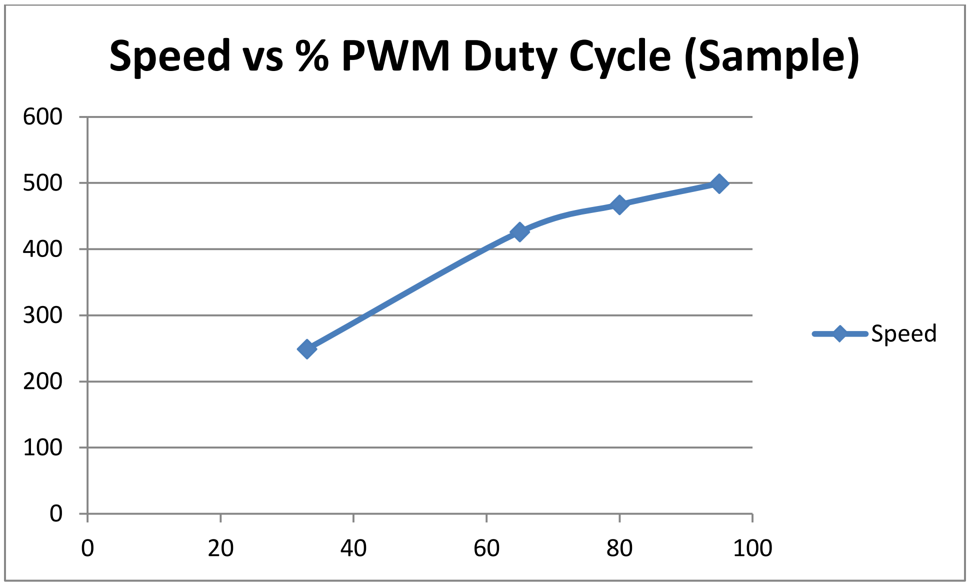


Figure 3. Sample Plot of DC motor speed as a function of PWM duty cycle

Appendix A: Project 10 Parts Configuration

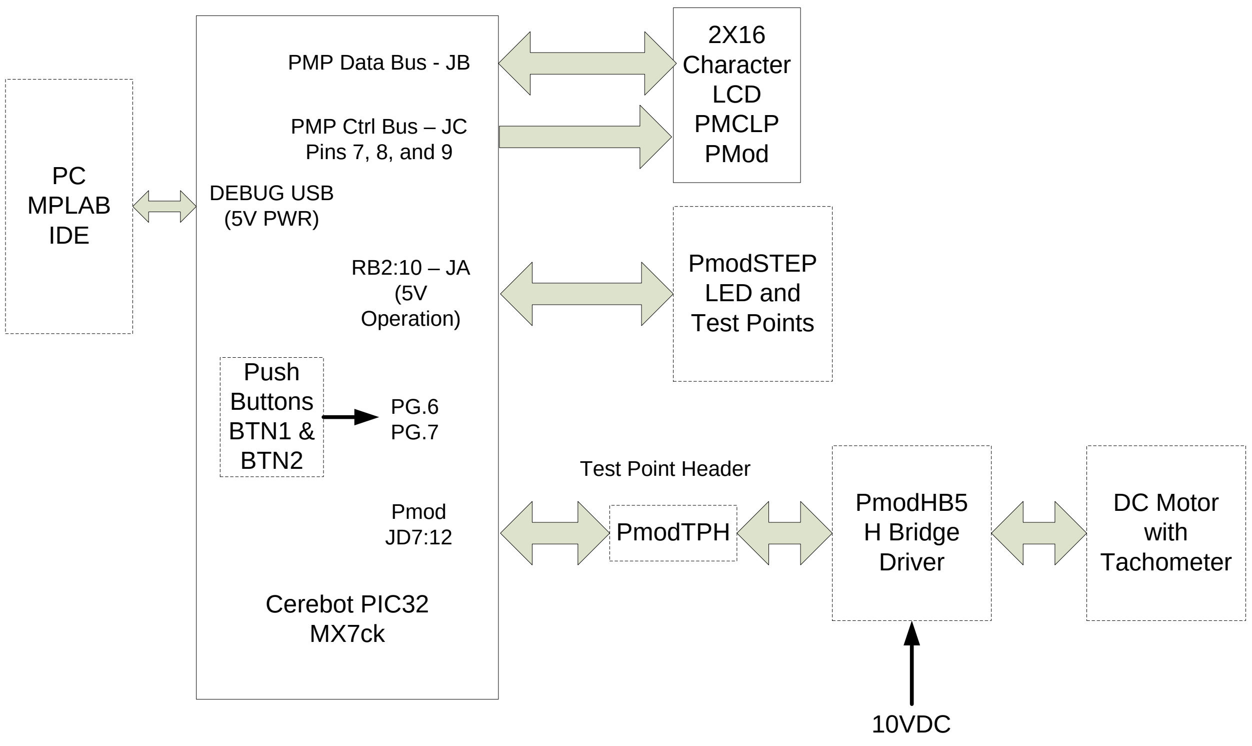


Figure 4. Block diagram of the equipment used in Project 7.

Appendix B: Motor Controller Wiring Diagram

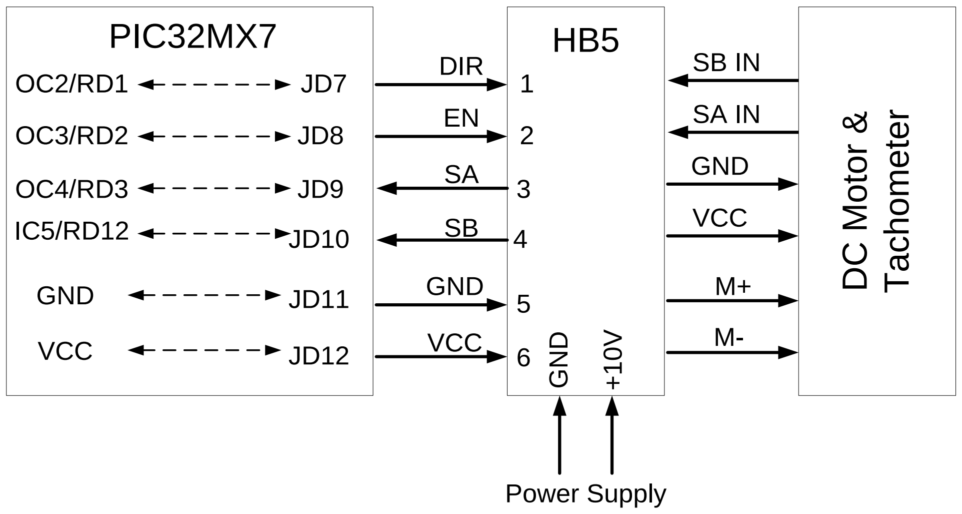
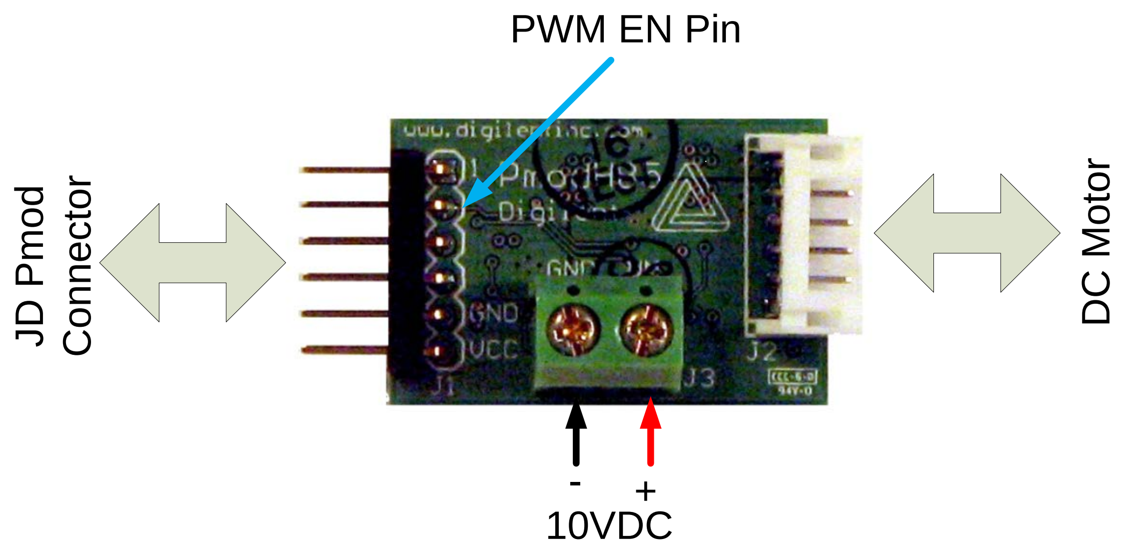


Figure 5. Motor connection diagram

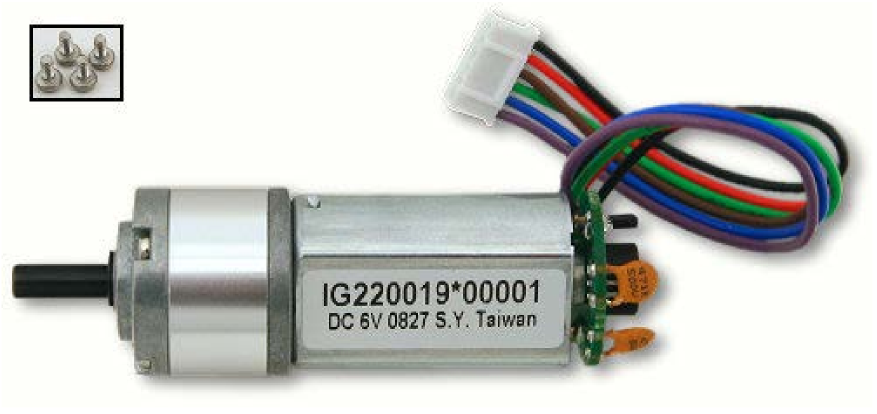
Appendix C: PMod HB5 Half H-Bridge Drive



Digilent PModHB5

Figure 6. PmodHB5 instrumentation connections

Appendix D: Geared DC Motor



Geared DC Motor with Tachometer

Figure 7. Digilent DC Motor

Appendix E: PmodCLP

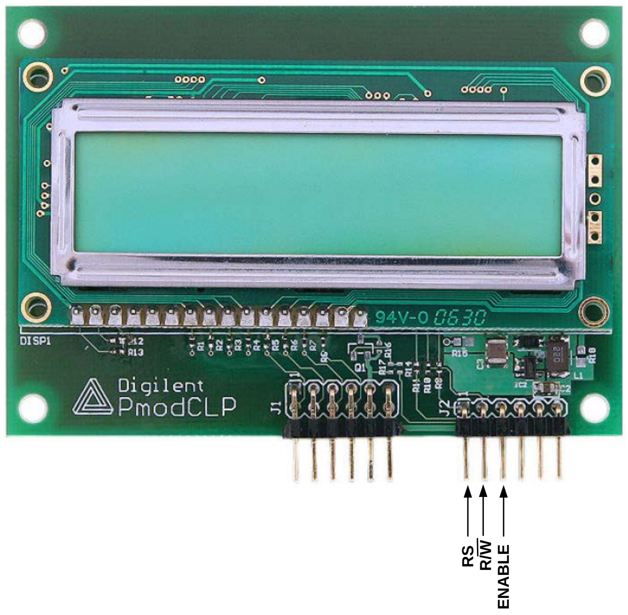


Figure 8. PmodCLP Character LCD pin identification

Appendix F: Period Measurement Resolution

Definition: *The granularity of a sensor measurement is the smallest change it can detect in the quantity that it is measuring. Resolution, which is the inverse of granularity, is related to the precision with which the measurement is made.*

Units: *Hz/Count*

Assume that Timer 3 count is recorded on each transition. Then for two frequencies, F1 and F2 the Timer 3 counts are computed by:

$$F_{TMR3} = \frac{PBCLK}{T3_{PS}} = \frac{10^7}{256} \qquad \text{Eq. 2}$$

$$T3_{COUNT1} = \frac{\mathrm{F_{TMR3}}}{\mathrm{F1}} \qquad \text{Eq. 3}$$

$$T3_{COUNT2} = \frac{\mathrm{F_{TMR3}}}{\mathrm{F2}} \qquad \text{Eq. 4}$$

To compute the resolution in Hz per count, assume that the count difference is unity which results in Eq. 5 or Eq. 6 which ever express the desired units.

$$|T3_{COUNT1} - T3_{COUNT2}| = \ Granularity = (F_{TMR3}) \cdot \left|\left(\frac{1}{\mathrm{F2}} - \frac{1}{\mathrm{F1}}\right)\right| COUNTS/\mathrm{Hz} \qquad \text{Eq. 5}$$

$$Resolution = \left(\frac{1}{\mathrm{F_{TMR3}}}\right) \cdot \left(\frac{1}{\left|\left(\frac{1}{\mathrm{F1}} - \frac{1}{\mathrm{F2}}\right)\right|}\right) = \left(\frac{1}{\mathrm{F_{TMR3}}}\right) \cdot \left(\frac{\mathrm{F1 \cdot F2}}{|(\mathrm{F2} - \mathrm{F1})|}\right) Hz/COUNT \qquad \text{Eq. 6}$$

$$|F2 - F1| = \mathrm{Resolution} = \left(\frac{\mathrm{F1 \cdot F2}}{\mathrm{F_{TMR3}}}\right) Hz/COUNT \qquad \text{Eq. 7}$$

If F1 approximately equals F2, at 1 Hz, the resolution is 0.0001024 Hz/ Count. At 200 Hz, the resolution is 1.024 Hz/ count.